UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/683,929	10/09/2003	John W. Rapp	1934-13-3	2222

7590 10/19/2006

Bryan A. Santarelli
GRAYBEAL JACKSON HALEY LLP
Suite 350
155 - 108th Avenue NE
Bellevue, WA 98004-5901

EXAMINER	
HUISMAN, DAVID J	

ART UNIT	PAPER NUMBER
2183	

DATE MAILED: 10/19/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	10/683,929	RAPP ET AL.	
	Examiner	Art Unit	
	David J. Huisman	2183	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *09 October 2003*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-65* is/are pending in the application.

4a) Of the above claim(s) *17-40 and 51-65* is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-16 and 41-50* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *11 March 2004* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date *4/11/05 & 9/19/05*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-16 and 41-50 have been examined.

Papers Submitted

2. It is hereby acknowledged that the following papers have been received subsequent to filing and placed of record in the file: Drawings as received on 3/11/2004, IDS as received on 4/11/2005, IDS as received on 9/19/2005, and Response to Restriction Requirement as received on 8/3/2006.

Information Disclosure Statement

3. The information disclosure statement filed on April 11, 2005, fails to fully comply with 37 CFR 1.98(a)(3) because it does not include a concise explanation of the relevance, as it is presently understood by the individual designated in 37 CFR 1.56(c) most knowledgeable about the content of the information, of the non-patent literature document entitled "Un Seul FPGA Dope Le Traitement D'Images" by Lecurieux-Lafayette, which is not in the English language. Consequently, this document has not been considered.

Election/Restrictions

4. Applicant's election with traverse of claims 1-16 and 41-50 (group I) in the reply filed on August 3, 2006, is acknowledged. The traversal is on the ground(s) that the examiner should consider restricting the remaining claims into fewer than six groups in the event that applicant decides to file one or more divisional applications. While fully considered, this is not found

persuasive because the examiner has determined that each group is directed to a different invention. Restriction ensures that only one invention be claimed per patent application. If the examiner were to combine any of the remaining six groups, then a divisional filed with any combination of the remaining groups would include multiple inventions.

The requirement is still deemed proper and is therefore made FINAL.

Specification

5. The title of the invention is not descriptive. A new title is required that is clearly indicative of the invention to which the claims are directed.

6. The lengthy specification has not been checked to the extent necessary to determine the presence of all possible minor errors. Applicant's cooperation is requested in correcting any errors of which applicant may become aware in the specification.

7. For each U.S. Patent application referred to in the specification, please remove the attorney docket number and insert the assigned serial number or the patent number, if the application has been issued.

Double Patenting

8. The nonstatutory double patenting rejection is based on a judicially created doctrine grounded in public policy (a policy reflected in the statute) so as to prevent the unjustified or improper timewise extension of the "right to exclude" granted by a patent and to prevent possible harassment by multiple assignees. A nonstatutory obviousness-type double patenting rejection is appropriate where the conflicting claims are not identical, but at least one examined application claim is not patentably distinct from the reference claim(s) because the examined application claim is either anticipated by, or would have been obvious over, the reference claim(s). See, e.g., *In re Berg*, 140 F.3d 1428, 46 USPQ2d 1226 (Fed. Cir. 1998); *In re Goodman*, 11 F.3d 1046, 29 USPQ2d 2010 (Fed. Cir. 1993); *In re Longi*, 759 F.2d 887, 225 USPQ 645 (Fed. Cir. 1985); *In re Van Ornum*, 686 F.2d 937, 214 USPQ 761 (CCPA 1982); *In re*

Vogel, 422 F.2d 438, 164 USPQ 619 (CCPA 1970); and *In re Thorington*, 418 F.2d 528, 163 USPQ 644 (CCPA 1969).

A timely filed terminal disclaimer in compliance with 37 CFR 1.321(c) or 1.321(d) may be used to overcome an actual or provisional rejection based on a nonstatutory double patenting ground provided the conflicting application or patent either is shown to be commonly owned with this application, or claims an invention made as a result of activities undertaken within the scope of a joint research agreement.

Effective January 1, 1994, a registered attorney or agent of record may sign a terminal disclaimer. A terminal disclaimer signed by the assignee must fully comply with 37 CFR 3.73(b).

9. Claims 1-2, 5-8, and 41-43 are provisionally rejected on the ground of nonstatutory obviousness-type double patenting as being unpatentable over claims 2, 2, 2, 2, 3, 3, 2, 3, and 3, respectively, of copending Application No. 10/683932 ('932). Although the conflicting claims are not identical, they are not patentably distinct from each other because:

a) Regarding claims 1 and 41 of the instant application, claim 2 of '932 anticipates (i.e., contains every element of) claims 1 and 41 of the instant application except for that claim 2 of '932 calls for driving the processed data onto the communication bus whereas claims 1 and 41 of the instant application call for providing the processed data to an external source. However, Official Notice is taken that a pipeline accelerator (i.e., a coprocessor) is known to be coupled to a processor and that the processor and coprocessor are coupled via "communication bus" so that they can communicate. This allows the coprocessor to relieve the processor of some work and communicate the results back to the processor, and also for the processor to send data to the coprocessor. Together, both components work to speed up execution and increase throughput (by performing more work per unit time together than a single unit would alone). Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 2 of '932 such that the communication bus is in fact coupling a processor and

pipeline accelerator so that processed data would be sent from the accelerator to the processor (i.e., external source).

b) Regarding claim 2 of the instant application, claim 2 of '932 has taught a pipeline accelerator as described in claim 1 of the instant application. Claim 2 of '932 has not taught that the memory is disposed on a first integrated circuit and the pipeline circuit is disposed on a second integrated circuit. However, Official Notice is taken that a pipeline circuit (found in a coprocessor or processor, for example) and memory (such as RAM) are well known to be disposed on separate integrated circuits. A person of ordinary skill in the art would have recognized that by disposing the two components on different chips, increased flexibility is realized. For example, if one chip goes bad, it may be replaced without having to replace the other. Or, if an upgrade to one of the components is desired, the chip may be swapped out for an upgraded chip without affecting the other. For instance, with multiple chips, upgrading the amount of memory in a system does not have to include upgrading the processor as well. In addition, as shown in *Nerwin v. Erlichman 168 USPQ 177 (1969)*, to make separable (i.e., to turn a single chip into two chips) is generally not given patentable weight or would have been an obvious improvement. As a result, in order to increase flexibility, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 2 of '932 such that the memory and pipeline circuit are disposed on separate integrated circuits.

c) Regarding claim 5 of the instant application, claim 2 of '932 has taught a pipeline accelerator as described in claim 1 of the instant application. Claim 2 of '932 has not taught that the external source comprises a processor and that the pipeline circuit is operable to receive the data from the processor. However, as described above, Official Notice is taken that a pipeline accelerator (i.e.,

coprocessor) is known to be coupled to a processor and that the processor and coprocessor are coupled such that they send data to and receive data from one another. One common occurrence is for a processor to send work to the coprocessor in the form of instruction data and other data so that it may perform work, thereby relieving the processor of some work. Together, both components work to speed up execution and increase throughput (by performing more work per unit time together than a single unit would alone). Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 2 of '932 such that the external source comprises a processor and that the pipeline circuit is operable to receive the data from the processor.

d) Regarding claim 6 of the instant application, claim 2 of '932 anticipates (i.e., contains every element of) claim 6 of the instant application except for that claim 2 of '932 calls for driving the processed data onto the communication bus whereas claim 6 of the instant application calls for providing the processed data to a processor, which is coupled to the pipeline accelerator. However, Official Notice is taken that a pipeline accelerator (i.e., a coprocessor) is known to be coupled to a processor and that the processor and coprocessor are coupled via "communication bus" so that they can communicate. This allows the coprocessor to relieve the processor of some work and communicate the results back to the processor, and also for the processor to send data to the coprocessor. Together, both components work to speed up execution and increase throughput (by performing more work per unit time together than a single unit would alone). Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 2 of '932 such that the communication bus is in fact coupling a

processor and pipeline accelerator so that processed data would be sent from the accelerator to the processor.

e) Regarding claims 7, 42, and 43 of the instant application, claim 3 of '932 anticipates (i.e., contains every element of) claims 7, 42, and 43 of the instant application except for that claim 3 of '932 claims a circuit is operable to load the retrieved data onto the communication bus whereas claims 7, 42, and 43 of the instant application claim a circuit operable to provide the retrieved processed data to an external source. However, Official Notice is taken that a pipeline accelerator (i.e., a coprocessor) is known to be coupled to a processor and that the processor and coprocessor are coupled via "communication bus" so that they can communicate. This allows the coprocessor to relieve the processor of some work and communicate the results back to the processor, and also for the processor to send data to the coprocessor. Together, both components work to speed up execution and increase throughput (by performing more work per unit time together than a single unit would alone). Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 3 of '932 such that the communication bus is in fact coupling a processor and pipeline accelerator so that processed data would be sent from the accelerator to the processor (i.e., external source).

f) Regarding claim 8 of the instant application, claim 3 of '932 anticipates (i.e., contains every element of) claim 8 of the instant application except for that claim 3 of '932 claims a circuit is operable to load the retrieved data onto the communication bus whereas claim 8 of the instant application claims a circuit operable to provide the retrieved processed data to a processor, which is coupled to the circuit. However, Official Notice is taken that a pipeline accelerator (i.e., a coprocessor) is known to be coupled to a processor and that the processor and coprocessor are

coupled via "communication bus" so that they can communicate. This allows the coprocessor to relieve the processor of some work and communicate the results back to the processor, and also for the processor to send data to the coprocessor. Together, both components work to speed up execution and increase throughput (by performing more work per unit time together than a single unit would alone). Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 3 of '932 such that the communication bus is in fact coupling a processor and pipeline accelerator so that processed data would be sent from the accelerator to the processor.

This is a provisional obviousness-type double patenting rejection because the conflicting claims have not in fact been patented.

10. Claim 3 is provisionally rejected on the ground of nonstatutory obviousness-type double patenting as being unpatentable over claim 2 of copending Application No. 10/683932 in view of the Free Online Dictionary of Computing, 1997 (herein referred to as FOLDOC).

11. Regarding claim 3 of the instant application, claim 2 of '932 has taught a pipeline accelerator as described in claim 1 of the instant application. Claim 2 of '932 has not taught that the pipeline circuit is disposed on a field-programmable gate array (FPGA). However, FOLDOC has taught that FPGAs are devices which are programmed after manufacture time. See the 1st paragraph of FOLDOC. This is clearly an advantage as designers are able to reprogram/reconfigure the device after it has been made. That is, if any new functionality were desired, a new chip would not have to be manufactured. Instead, the associated functionality would simply be programmed into the FPGA. Consequently, in order to make the pipeline

circuit of claim 2 of '932 reconfigurable and, consequently, more flexible, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 2 of '932 such that the pipeline is disposed on an FPGA as taught by FOLDOC.

This is a provisional obviousness-type double patenting rejection.

12. Claims 4, 9-10, 16, 44-45, and 49-50 are provisionally rejected on the ground of nonstatutory obviousness-type double patenting as being unpatentable over claim 2 of copending Application No. 10/683932 in view of Morikawa et al., U.S. Patent No. 5,909,565 (herein referred to as Morikawa). In addition, FOLDOC is cited as extrinsic evidence for showing that "raw data" is equivalent to data.

13. Regarding claim 4 of the instant application, claim 2 of '932 has taught a pipeline accelerator as described in claim 1 of the instant application. Claim 2 of '932 has not taught the that the pipeline circuit is operable to provide the processed data to the external source by loading the processed data into the memory, retrieving the processed data from the memory, and providing the retrieved processed data to the external source. Essentially, claim 2 of '932 has not taught buffering the processed data before sending it to the external source. However, Morikawa has taught such a concept. See Fig.4 and note that the data, after being processed by processing unit 211, is stored in memory portion 231 (part of "the memory") before being retrieved from the buffer and sent to the external source 201. As is known in the art, buffering provides the advantage of holding data until something can be done with it. For instance, the data may not be able to be sent to the external source at time X due to bus contention, and consequently, if it is not buffered, the component producing that data will have to stall. As a result, in order to reduce

stalling, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 2 of '932 such that the pipeline circuit is operable to provide the processed data to the external source by loading the processed data into the memory, retrieving the processed data from the memory, and providing the retrieved processed data to the external source.

14. Regarding claim 9 of the instant application, claim 2 of '932 has taught a pipeline accelerator comprising:

a) a first memory. See claim 2, line 3, of '932, and note "a respective memory".

b) a hardwired-pipeline circuit coupled to the memory. See claim 2, line 3, of '932.

c) the hardwired-pipeline circuit comprising an input-data handler operable to receive raw data and to load the raw data into the first memory. See claim 2, lines 4-6, of '932, and note that according to FOLDOC, "raw data" is equivalent to data. Furthermore, if claim 2 of '932 is performing receiving and loading, then some inherent component is actually performing those functions, and that component is the "input-data handler."

d) the hardwired-pipeline circuit comprising a hardwired pipeline operable to process the raw data. See claim 2, lines 4 and 8, of '932.

e) a pipeline interface operable to retrieve the raw data from the first memory and provide the retrieved raw data to the hardwired pipeline. See claim 2, lines 4 and 7-8, of '932. As above, if claim 2 of '932 is performing retrieving and providing to the pipeline, then some inherent component is actually performing those functions, and that component is the "pipeline interface."

f) an output-data handler operable to provide the processed data. See the last line of claim 2 of '932. As above, if claim 2 of '932 is performing providing the processed data, then some

inherent component is actually performing that function, and that component is the "output data handler."

g) claim 2 of '932 has not taught loading raw data from an external source and providing processed data to the external source. However, Morikawa has taught such a concept. See Fig.4 and first note that coprocessor 202 communicates with external source (processor 201) by receiving data via bus 124 and sending data via bus 125. It is common for coprocessors to be coupled to processors so that they may work together to solve a task. This results in increased throughput as two units may perform more work together than individually. And, the communication between both processor and coprocessor allows for results and instructions to be passed between them. As a result, it would have been obvious to one of ordinary skill in the art at he time of the invention to modify claim 2 of '932 in view of Morikawa such that raw data is loaded from an external source and processed data is provided to the external source. This allows for communication between two components working together to speed up execution.

h) claim 2 of '932 has not taught a second memory, where the hardwired pipeline is coupled to the second memory, the pipeline interface loads processed data from the hardwired pipeline into the second memory, and an output-data handler operable to retrieve the processed data from the second memory before providing the processed data. Essentially, claim 2 of '932 has not taught buffering the processed data before providing it. However, Morikawa has taught such a concept. See Fig.4 and note that the data, after being processed by processing unit 211, is stored in memory portion 231 before being retrieved from the buffer and provided as an output to the external source 201. As described above, if retrieving from a second memory and providing data is performed, then some inherent component must be performing those functions. That

component is the "output data handler". As is known in the art, buffering provides the advantage of holding data until something can be done with it. For instance, the data may not be able to be sent to the external source at time X due to bus contention, and consequently, if it is not buffered, the component producing that data will have to stall. As a result, in order to reduce stalling, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 2 of '932 such that the pipeline circuit includes a second memory coupled to the second memory, the pipeline interface loads processed data from the hardwired pipeline into the second memory, and an output-data handler operable to retrieve the processed data from the second memory.

15. Referring to claim 10, claim 2 of '932 in view of Morikawa has taught a pipeline accelerator as described in claim 9.

a) Claim 2 of '932 has not taught that the first memory includes first and second ports, the input-data handler is operable to load the raw data via the first port of the first memory, and the pipeline interface is operable to retrieve the raw data via the second port of the first memory. However, Morikawa has taught these concepts. See Fig.4 and note that the first memory 230 includes an input port and an output port, the first memory 230 receives data via its input port (coupled to driver 251 and bus 124, and that the data is read from memory 230 via its output port which sends data to processing unit 211. Clearly, multiple ports are advantageous because the more ports that are available, the more accesses to memory can occur at a given time. With a single port, only one read/write may occur at once, whereas with two ports (input and output), a read and write could occur simultaneously, which speeds up the system. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim

2 of'932 such that the first memory includes first and second ports, the input-data handler is operable to load the raw data via the first port of the first memory, and the pipeline interface is operable to retrieve the raw data via the second port of the first memory.

b) Morikawa has further taught that:

b1) the second memory includes first and second ports. See Fig.4 and note the I/O ports of second memory 231.

b2) the pipeline interface is operable to load the processed data via the first port of the second memory. After the data is processed by unit 211 of Fig.4, the processed data is written to the second memory 231 via its input port.

d) the output-data handler is operable to retrieve the processed data via the second port of the second memory. See Fig.4 and note that when the data is to be sent to the processor 201, it is first retrieved from memory 231 via its output port.

16. Referring to claim 16, claim 2 of '932 in view of Morikawa has taught a method as described in claim 9.

a) claim 2 of '932 in view of Morikawa has inherently taught that each of the input-data handler, hardwired pipeline, pipeline interface, and output-data handler has a respective operating status. That is, at the very least, each component in the system is either operating or not operating (and these are statuses).

b) claim 2 of '932 in view of Morikawa has not taught an exception manager coupled to and operable to identify an exception in the input-data handler, hardwired pipeline, pipeline interface, or output-data handler in response to the operating statuses. However, Official Notice is taken that checking for errors in a pipeline during processing is well known and accepted in the art.

For instance, if a pipeline unit is performing a divide operation, which is a well known operation performed by pipeline units, then a division of a number by 0 should raise an exception, as it is an illegal operation. This type of error should be monitored so that the system can take appropriate action to correct the error. As a result, in order to ensure proper execution, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 2 of '932 in view of Morikawa such that the pipeline unit of claim 2 of '932 in view of Morikawa performs divide operations (in order to obtain the ability to perform division) and exceptions are detected in a hardwired pipeline.

17. Regarding claim 44 of the instant application, claim 2 of '932 has taught a method comprising:

a) loading raw data into a first memory. See claim 2, line 6, of '932. Also, please see FOLDOC and note that "raw data" is equivalent to "data."

b) retrieving the raw data from the first memory. See claim 2, line 7, of '932.

c) processing the retrieved data with a hardwired pipeline. See claim 2, lines 4 and 8, of '932.

d) providing the processed data. See the last line of claim 2 of '932.

e) claim 2 of '932 has not taught loading raw data from an external source and providing data to an external source. However, Morikawa has taught such a concept. See Fig.4 and first note that coprocessor 202 communicates with external source (processor 201) by receiving data via bus 124 and sending data via bus 125. It is common for coprocessors to be coupled to processors so that they may work together to solve a task. This results in increased throughput as two units may perform more work together than individually. And, the communication between both processor and coprocessor allows for results and instructions to be passed between them. As a

result, it would have been obvious to one of ordinary skill in the art at he time of the invention to modify claim 2 of '932 in view of Morikawa such that raw data is loaded from an external source and processed data is provided back to the external source. This allows for communication between two components working together to speed up execution.

f) claim 2 of '932 has not taught loading the processed data from the hardwired pipeline into a second memory, and providing processed data from the second memory to the external source. Essentially, claim 2 of '932 has not taught buffering the processed data before providing it to the external source. However, Morikawa has taught such a concept. See Fig.4 and note that data, after being processed by processing unit 211, is stored in memory portion 231 before being provided to the external source 201. As is known in the art, buffering provides the advantage of holding data until something can be done with it. For instance, the data may not be able to be sent to the external source at time X due to bus contention, and consequently, if it is not buffered, the component producing that data will have to stall. As a result, in order to reduce stalling, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 2 of '932 in view of Morikawa such that the pipeline circuit includes a second memory coupled to the second memory, the pipeline interface loads processed data from the hardwired pipeline into the second memory, and an output-data handler operable to retrieve the processed data from the second memory.

18. Referring to claim 45, claim 2 of '932 in view of Morikawa has taught a method as described in claim 44.

a) claim 2 of '932 has further taught loading the raw data comprises loading the raw data via a first port of the first memory. The examiner deems this to be inherent as a port must exist for data to be loaded.

b) claim 2 of '932 has not taught retrieving the raw data comprises retrieving the raw data via the second port of the first memory. However, Morikawa has taught a same first memory with multiple ports. See Fig.4 and note that the data is read from memory 230 via its output port and sent to the coprocessor processing unit 211. Clearly, multiple ports are advantageous because the more ports that are available, the more accesses to memory can occur at a given time. With a single port, only one read/write may occur at once, whereas with two ports (input and output), a read and write could occur simultaneously, which speeds up the system. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 2 of'932 such that the first memory includes a second port from which raw data is read.

c) Morikawa has further taught loading the processed data comprises loading the processed data via a first port of the second memory and providing the processed data comprises retrieving the processed data via a second port of the second memory. See Fig.4 and note that the processed data is written to the second memory 231 via its input port and when the data is to be sent to the processor 201, it is first retrieved from memory 231 via its output port.

19. Referring to claim 49, claim 2 of '932 in view of Morikawa has taught a method as described in claim 44. Furthermore, claim 2 of '932 in view of Morikawa has inherently taught setting parameters for loading and retrieving the raw data, processing the retrieved data, and loading and providing the processed data. That is, if the system is going to load data from the processor, process the data, and then provide data back to the processor, then the system must set

which data will be loaded (i.e., from where it will be loaded), it must set which operation is to be performed on the data, and it must set which destination will be receiving the processed data result. Without these parameters, the functions could not be performed.

20. Referring to claim 50, claim 2 of '932 in view of Morikawa has taught a method as described in claim 44. While claim 2 of '932 in view of Morikawa has not explicitly taught determining whether an error occurs during the loading and retrieving of the raw data, the processing of the retrieved data, and the loading and providing of the processed data, Official Notice is taken that checking for errors during processing is well known and accepted in the art. For instance, if a pipeline is performing a divide operation, which is a well known operation performed by pipelines, then a division of a number by 0 should raise an error as it is an illegal operation. This type of error should be monitored so that the system can take appropriate action to correct the error. As a result, in order to ensure proper execution, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 2 of '932 in view of Morikawa such that the pipeline of claim 2 of '932 in view of Morikawa performs divide operations (in order to obtain the ability to perform the common operation of division) and errors are detected during the loading and retrieving of the raw data, the processing of the retrieved data, and the loading and providing of the processed data.

This is a provisional obviousness-type double patenting rejection.

21. Claims 11 and 46 are provisionally rejected on the ground of nonstatutory obviousness-type double patenting as being unpatentable over claim 2 of copending Application No.

10/683932 in view of Morikawa and further in view of Fette et al., U.S. Patent No. 4,862,407 (herein referred to as Fette).

22. Referring to claim 11, claim 2 of '932 in view of Morikawa has taught a pipeline accelerator as described in claim 9. Claim 2 of '932 in view of Morikawa has not taught a third memory coupled to the hardwired-pipeline circuit, wherein the hardwired pipeline is operable to generate intermediate data while processing the raw data, and wherein the pipeline interface is operable to load the intermediate data into the third memory and to retrieve the intermediate data from the third memory. However, Fette has taught such a concept. Specifically, Fette has taught that coprocessors (i.e., pipeline accelerators) contain multiply-accumulate circuitry. See the last sentence of the abstract. A multiply-accumulator is known to include an accumulation register (third memory). With a multiply-accumulate (MAC) operation, values (raw data) are multiplied to generate a multiplication result. This result is then added to the value already in the accumulation register to generate intermediate data. This intermediate data is then stored in the third memory (back into the accumulator), where it is later retrieved and added to the next multiplication result. A MAC operation is a common operation, and it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 2 of '932 in view of Morikawa such that the coprocessor pipeline performs MAC operations and therefore includes the claimed third memory and its purpose. One would have been motivated to make such a combination to increase the functionality of the processing portion by giving it MAC functionality.

23. Referring to claim 46, claim 2 of '932 in view of Morikawa has taught a method as described in claim 44. Claim 2 of '932 in view of Morikawa has not taught generating

intermediate data with the hardwired pipeline in response to processing the raw data, loading the intermediate data into a third memory, and providing the intermediate data from the third memory back to the hardwired pipeline. However, Fette has taught such a concept. Specifically, Fette has taught that coprocessors (i.e., pipeline accelerators) contain multiply-accumulate circuitry. See the last sentence of the abstract. A multiply-accumulator is known to include an accumulation register (third memory). With a multiply-accumulate (MAC) operation, values (raw data) are multiplied to generate a multiplication result. This result is then added to the value already in the accumulation register to generate intermediate data. This intermediate data is then stored in the third memory (back into the accumulator), where it is later retrieved and added to the next multiplication result. A MAC operation is a common operation, and it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 2 of '932 in view of Morikawa such that the coprocessor pipeline performs MAC operations and therefore includes the claimed third memory and its purpose. One would have been motivated to make such a combination to increase the functionality of the coprocessor by giving it MAC functionality.

This is a provisional obviousness-type double patenting rejection.

24. Claims 12 and 15 are provisionally rejected on the ground of nonstatutory obviousness-type double patenting as being unpatentable over claim 2 of copending Application No. 10/683932 in view of Morikawa and further in view of FOLDOC.

25. Referring to claim 12, claim 2 of '932 in view of Morikawa has taught a pipeline accelerator as described in claim 9.

a) claim 2 of '932 in view of Morikawa has not taught that the first and second memories are respectively disposed on first and second integrated circuits. However, as shown in *Nerwin v. Erlichman 168 USPQ 177 (1969)*, to make separable is generally not given patentable weight or would have been an obvious improvement. In this case, it would have been obvious to have the buffers of the coprocessor of Fig.4 (of Morikawa) to be on separate chips. For instance, everything prior to the processing unit 211 would be on one chip, while the rest would be on another. A person of ordinary skill in the art would have recognized that separating the components into multiple chips allows for more flexibility and cost savings. For example, if the coprocessor were divided into multiple chips and one chip is malfunctioning or is defective, just that chip would need to be replaced while the other functioning chip would be left in tact. This results in not having to spend money replacing the functioning chip. Another advantage would be to gain the ability to upgrade a portion of the coprocessor. As a result, in order to increase flexibility, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 2 of '932 in view of Morikawa to include the first and second memories on separate integrated circuits.

b) claim 2 of '932 in view of Morikawa has not taught that the pipeline circuit is disposed on a field-programmable gate array (FPGA). However, FOLDOC has taught that FPGAs are devices which are programmed after manufacture time. See the 1st paragraph of FOLDOC. This is clearly an advantage as designers are able to reprogram/reconfigure the device after it has been made. That is, if any new functionality is desired, a new chip would not have to be manufactured. Instead, the associated functionality would simply be programmed into the FPGA. Consequently, in order to make the pipeline circuit of claim 2 of '932 in view of

Morikawa reconfigurable and, consequently, more flexible, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 2 of '932 in view of Morikawa such that the pipeline is disposed on an FPGA, as taught by FOLDOC.

26. Referring to claim 15, claim 2 of '932 in view of Morikawa has taught a pipeline accelerator as described in claim 9.

a) claim 2 of '932 in view of Morikawa has further taught that each of the input-data handler, hardwired pipeline, pipeline interface, and output-data handler has a respective operating configuration. That is, each component in claim 2 of '932 in view of Morikawa operates in a specific fashion and consequently, each component inherently has a respective operating configuration.

b) claim 2 of '932 in view of Morikawa has not taught a configuration manager coupled to and operable to set the operating configurations of the input-data handler, hardwired pipeline, pipeline interface, and output-data handler. However, FOLDOC has taught that FPGAs are devices which are programmed after manufacture time. See the 1st paragraph of FOLDOC. This is clearly an advantage as designers are able to reprogram/reconfigure the device after it has been made. That is, if any new functionality is desired, a new chip would not have to be manufactured. Instead, the associated functionality would simply be programmed into the FPGA. Consequently, in order to make the pipeline circuit of claim 2 of '932 in view of Morikawa reconfigurable and, consequently, more flexible, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 2 of '932 in view of Morikawa such that the system is disposed on an FPGA. Furthermore, an FPGA would be

coupled to a configuration manager so that the FPGA may be programmed to include the desired functionality.

This is a provisional obviousness-type double patenting rejection.

27. Claims 13-14 and 47-48 are provisionally rejected on the ground of nonstatutory obviousness-type double patenting as being unpatentable over claim 2 of copending Application No. 10/683932 in view of Morikawa and further in view of Frey et al., U.S. Patent No. 5,185,871 (herein referred to as Frey).

28. Referring to claim 13, claim 2 of '932 in view of Morikawa has taught a pipeline accelerator as described in claim 9. Claim 2 of '932 in view of Morikawa has not taught an input-data queue coupled to the input-data handler and the pipeline interface, wherein the input-data handler is operable to load into the input-data queue a pointer to a location of the raw data within the first memory and wherein the pipeline interface is operable to retrieve the raw data from the location using the pointer. However, Frey has taught such a concept. See Fig.3, components 17 and 21, and column 11, line 67, to column 12, line 5. Essentially, operand addresses in a multi-operand buffer are stored in an operand fetch queue so that when it is time to fetch the operands, they are located appropriately. A person of ordinary skill in the art would have recognized that by implementing a multi-operand buffer in claim 2 of '932 in view of Morikawa (for the first memory/buffer 230), multiple operands would be buffered at a time, which ensures that the coprocessor always has enough work to do. Also, if the coprocessor were to stall in any way, the multi-operand buffer would still be able to buffer operands from the processor, thereby allowing the processor to continue on with other work. With such a system,

saving the pointer of an operand allows for the location of the operand in the buffer. Clearly, when an instruction is to operate on an operand, the correct operand should be located, and so the location of the operand must be remembered. Therefore, in order to buffer more operands, ensure work, and avoid stalls, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 2 of '932 in view of Morikawa to include a multi-operand input buffer and an input-data queue for holding a pointer to an operand (raw data) within the buffer and then using the pointer to locate the desired operand.

29. Referring to claim 14, claim 2 of '932 in view of Morikawa has taught a pipeline accelerator as described in claim 9. Claim 2 of '932 in view of Morikawa has not taught an output-data queue coupled to the output-data handler and the pipeline interface wherein the pipeline interface is operable to load into the output-data queue a pointer to a location of the processed data within the second memory and wherein the output-data handler is operable to retrieve the processed data from the location using the pointer. However, Frey has taught keeping a queue of pointers so that data in a multiple-entry buffer would be appropriately located. See Fig.3, components 17 and 21, and column 11, line 67, to column 12, line 5. A person of ordinary skill in the art would have recognized that by implementing a multi-result buffer in claim 2 of '932 in view of Morikawa (for the second memory/buffer 231), multiple results would be buffered at a time, which ensures that the coprocessor would function smoothly if bus contention were present. For instance, the regular processor may be trying to write to its own register file. If this is the case, the coprocessor cannot write a result to the register file at the same time. Consequently, if only a single entry output buffer exists, then the result must be stored in that buffer until the register file is available. Until its available, another result cannot

be produced as the previous result is already in the buffer. This would result in stalling the coprocessor. With a multiple-entry buffer, the coprocessor would be able to continue executing and storing results until the register file becomes available. And, with such a system, saving the pointer of the result allows for the location of the result in the buffer. Clearly, when a result is to be written to the register file, the correct result should be written, and so the location of the result must be remembered. Therefore, in order to avoid stall potential, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 2 of '932 in view of Morikawa to include a multiple-entry output buffer and an output-data queue for holding a pointer to a result (processed data) within the buffer and then using the pointer to locate the desired data.

30. Referring to claim 47, claim 2 of '932 in view of Morikawa has taught a method as described in claim 44. Claim 2 of '932 in view of Morikawa has not taught loading into an input-message queue a pointer to a location of the raw data within the first memory, and wherein retrieving the raw data comprises retrieving the raw data from the location using the pointer. However, Frey has taught such a concept. See Fig.3, components 17 and 21, and column 11, line 67, to column 12, line 5. Essentially, addresses of operands in a multi-operand buffer are stored in an operand fetch queue so that when it is time to fetch the operands, they are located appropriately. A person of ordinary skill in the art would have recognized that by implementing a multi-operand buffer in claim 2 of '932 in view of Morikawa (for the first memory/buffer 230), multiple operands would be buffered at a time, which ensures that the coprocessor always has enough work to do. Also, if the coprocessor were to stall in any way, the multi-operand buffer would still be able to buffer operands from the processor, thereby allowing the processor to

continue on with other work. With such a system, saving the pointer of an operand allows for the location of the operand in the buffer. Clearly, when an instruction is to operate on an operand, the correct operand should be located, and so the location of the operand must be remembered. Therefore, in order to buffer more operands, ensure work, and avoid stalls, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 2 of '932 in view of Morikawa to include a multi-operand input buffer and an input-message queue for holding a pointer to an operand (raw data) within the buffer and then using the pointer to locate the desired operand.

31. Referring to claim 48, claim 2 of '932 in view of Morikawa has taught a method as described in claim 44. Claim 2 of '932 in view of Morikawa has not taught loading into an output-message queue a pointer to a location of the processed data within the second memory, and wherein retrieving the processed data comprises retrieving the processed data from the location using the pointer. However, Frey has taught keeping a queue of pointers so that data in a multiple-entry buffer would be appropriately located. See Fig.3, components 17 and 21, and column 11, line 67, to column 12, line 5. A person of ordinary skill in the art would have recognized that by implementing a multi-result buffer in claim 2 of '932 in view of Morikawa (for the second memory/buffer 231), multiple results would be buffered at a time, which ensures that the coprocessor would function smoothly if bus contention were present. For instance, the regular processor may be trying to write to its own register file. If this is the case, the coprocessor cannot write a result to the register file at the same time. Consequently, if only a single entry output buffer exists, then the result must be stored in that buffer until the register file is available. Until its available, another result cannot be produced as the previous result is

already in the buffer. This would result in stalling the coprocessor. With a multiple-entry buffer, the coprocessor would be able to continue executing and storing results until the register file becomes available. With such a system, saving the pointer of the result allows for the location of the result in the buffer. Clearly, when a result is to be written to the register file, the correct result should be written, and so the location of the result must be remembered. Therefore, in order to avoid stall potential, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify claim 2 of '932 in view of Morikawa to include a multiple-entry output buffer and an output-message queue for holding a pointer to a result (processed data) within the buffer and then using the pointer to locate the desired data.

This is a provisional obviousness-type double patenting rejection.

Claim Rejections - 35 USC § 102

32. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

33. Claims 1, 4-10, 41-45, and 49 are rejected under 35 U.S.C. 102(b) as being anticipated by Morikawa.

34. Referring to claim 1 (under a first interpretation), Morikawa has taught a pipeline accelerator, comprising:

a) a memory. See Fig.4, and note that input buffer 230 and output buffer 231 make up a memory (an I/O buffer memory).

b) a hardwired-pipeline circuit (see Fig.4; the portions of coprocessor 202 excluding the buffers; also see the bottom portion of Fig.6 and note that the coprocessor is pipelined) coupled to the memory and operable to:

b1) receive data. See Fig.4 and claims 1-2 of Morikawa and note that data from register file 122 is received by the pipeline circuit via bus 124 and latch 118.

b2) load the data into the memory. Fig.4 shows that the register data coming from the main processor on bus 124 and latch 118 is ultimately put into the memory portion 230 by the driver 251 of the pipeline circuit.

b3) retrieve the data from the memory. Fig.4 shows that the data is retrieved from the memory portion 230 by the coprocessor processing unit 211.

b4) process the retrieved data. Once the data has been retrieved, it is processed by the processing unit. See column 18, lines 22-27.

b5) provide the processed data to an external source. See column 18, lines 28-32, and Fig.4. Note that after processing, the data is sent to the processor 201, which is external to the coprocessor (hardwired-pipeline circuit).

35. Referring to claim 4, Morikawa has taught a pipeline accelerator as described in claim 1. Morikawa has further taught that the pipeline circuit is operable to provide the processed data to the external source by:

a) loading the processed data into the memory. See Fig.4, and note that after the data is processed, it is stored in memory portion 231 through driver 255.

b) retrieving the processed data from the memory and providing the retrieved processed data to the external source. Again, see column 18, lines 28-32, and Fig.4. The data is retrieved from the memory portion 231 and sent to the processor via bus 125.

36. Referring to claim 5, Morikawa has taught a pipeline accelerator as described in claim 1. Morikawa has further taught that:

a) the external source comprises a processor. See Fig.4, component 201.

b) the pipeline circuit is operable to receive the data from the processor. See Fig.4. The data that is sent to the coprocessor (pipeline circuit) is from the processor.

37. Referring to claim 6, Morikawa has taught a computing machine comprising:

a) a processor. See Fig.4, component 201.

b) a pipeline accelerator (Fig.4, component 202) coupled to the processor and comprising:

b1) a memory. See Fig.4, and note that input buffer 230 and output buffer 231 make up a memory (an I/O buffer memory).

b2) a hardwired-pipeline circuit (see Fig.4; the portions of component 202 excluding the buffers; also see the bottom portion of Fig.6 and note that the coprocessor is pipelined) coupled to the memory and operable to:

- receive data from the processor. See Fig.4 and claims 1-2 of Morikawa and note that data from register file 122 (in the processor 201) is received by the pipeline circuit via bus 124 and latch 118.
- load the data into the memory. Fig.4 shows that the register data coming from the main processor on bus 124 and latch 118 is ultimately put into the memory portion 230 by the driver 251 of the pipeline circuit.

- retrieve the data from the memory. Fig.4 shows that the data is retrieved from the memory portion 230 by the coprocessor processing unit 211.
- process the retrieved data. Once the data has been retrieved, it is processed by the processing unit. See column 18, lines 22-27.
- provide the processed data to the processor. See column 18, lines 28-32, and Fig.4. Note that after processing, the data is sent to the processor on bus 125 via driver 257.

38. Referring to claim 7, Morikawa has taught a pipeline accelerator comprising:

a) a memory. See Fig.4, and note that input buffer 230 and output buffer 231 make up a memory (an I/O buffer memory).

b) a hardwired-pipeline circuit (see Fig.4; the portions of component 202 excluding the buffers; also see the bottom portion of Fig.6 and note that the coprocessor is pipelined) coupled to the memory and operable to:

b1) receive data. See Fig.4 and claims 1-2 of Morikawa and note that data from register file 122 is received by the pipeline circuit via bus 124 and latch 118.

b2) process the received data. After being received, it is processed by the coprocessor processing unit 211. See column 18, lines 22-27.

b3) load the processed data into the memory. See Fig.4, and note that after the data is processed, it is stored in memory portion 231.

b4) retrieve the processed data from the memory and provide the retrieved processed data to an external source. See column 18, lines 28-32, and Fig.4. The data is retrieved from the memory portion 231 and sent to the processor via driver 257 and bus 125.

39. Referring to claim 8, Morikawa has taught a computing machine comprising:

a) a processor. See Fig.4, component 201.

b) a pipeline accelerator (Fig.4, component 202) coupled to the processor and comprising:

b1) a memory. See Fig.4, and note that input buffer 230 and output buffer 231 make up a memory (an I/O buffer memory).

b2) a hardwired-pipeline circuit (see Fig.4; the portions of component 202 excluding the buffers; also see the bottom of portion Fig.6 and note that the coprocessor is pipelined) coupled to the memory and operable to:

- receive data from the processor. See Fig.4 and claims 1-2 of Morikawa and note that data from register file 122 (in the processor) is received by the pipeline circuit via bus 124 and latch 118.
- process the received data. After being received, it is processed by the coprocessor processing unit 211. See column 18, lines 22-27.
- load the processed data into the memory. See Fig.4, and note that after the data is processed, it is stored in memory portion 231.
- retrieve the processed data from the memory and provide the retrieved processed data to the processor. See column 18, lines 28-32, and Fig.4. The data is retrieved from the memory portion 231 and sent to the processor via driver 257 and bus 125.

40. Referring to claim 9, Morikawa has taught a pipeline accelerator comprising:

a) first and second memories. See Fig.4, components 230 and 231 (input and output buffers).

b) a hardwired-pipeline circuit (see Fig.4; the portions of component 202 excluding the buffers; also see the bottom portion of Fig.6 and note that the coprocessor is pipelined) coupled to the first and second memories and comprising:

b1) an input-data handler operable to receive raw data from an external source and to load the raw data into the first memory. See Fig.4 and claims 1-2 of Morikawa and note that data from register file 122 (in the processor) is received by the pipeline circuit via bus 124 and latch 118. Fig.4 also shows that the register data coming from the main processor on bus 124 is ultimately put into the first memory 230 by the driver 251 of the pipeline circuit. Furthermore, it should be noted that the portions of the system receiving, controlling receiving, and loading data make up the input data handler.

b2) a hardwired pipeline operable to process the raw data. See Fig.4, component 211 and the bottom portion of Fig.6.

b3) a pipeline interface operable to retrieve the raw data from the first memory. See Fig.4, and note that before the data is processed by unit 211, it must be retrieved from the first memory 230. The portion of the system retrieving the data would be the pipeline interface.

b4) provide the retrieved raw data to the hardwired pipeline. See Fig.4, and note that the date from the first memory 230 is provided to the hardwired pipeline 211.

b5) load processed data from the hardwired pipeline into the second memory. See Fig.4, and note that after the data is processed, it is loaded into output buffer 231.

b6) an output-data handler operable to retrieve the processed data from the second memory and to provide the processed data to the external source. See column 18, lines

28-32, and Fig.4. The data is retrieved from the memory portion 231 and sent to the processor via driver 257 and bus 125. The portion of the system controlling the outputting is the output data handler.

41. Referring to claim 10, Morikawa has taught a pipeline accelerator as described in claim 9. Morikawa has further taught that:

a) the first and second memories each include respective first and second ports. See Fig.4 and note that each of the memories 230 and 231 include an input and output port.

b) the input-data handler is operable to load the raw data via the first port of the first memory. See Fig.4 and note that data is loaded into first memory 230 via its input port.

c) the pipeline interface is operable to retrieve the raw data via the second port of the first memory and to load the processed data via the first port of the second memory. See Fig.4 and note that the data is read from memory 230 via its output port, it is processed by component 211, and then the processed data is written to the second memory 231 via its input port.

d) the output-data handler is operable to retrieve the processed data via the second port of the second memory. See Fig.4 and note that when the data is to be sent to the processor 201, it is first retrieved from memory 231 via its output port.

42. Referring to claim 41, Morikawa has taught a method comprising:

a) loading data into a memory. See Fig.4 and claims 1-2 of Morikawa and note that data from register file 122 is received by the coprocessor and loaded into memory portion 230 (input buffer).

b) retrieving the data from the memory. Fig.4 shows that the data is retrieved from the memory portion 230 by the coprocessor processing unit 211.

c) processing the retrieved data with a hardwired-pipeline circuit. Once the data has been retrieved, it is processed by the processing unit 211. See column 18, lines 22-27. Also, see the bottom portion of Fig.6 and note that the system is pipelined.

d) providing the processed data to an external source. See column 18, lines 28-32, and Fig.4. Note that after processing, the data is sent to the processor, which is external to the coprocessor (hardwired-pipeline circuit) via driver 257 and bus 125.

43. Referring to claim 42, Morikawa has taught a method as described in claim 41. Morikawa has further taught that providing the processed data to an external source comprises:

a) loading the processed data into the memory. See Fig.4, and note that after the data is processed, it is loaded into memory portion 231. Note that portions 230 and 231 make up a I/O buffer memory.

b) retrieving the processed data from the memory and providing the retrieved processed data to the external source. Again, see column 18, lines 28-32, and Fig.4. The data is retrieved from the memory portion 231 and sent to the processor via driver 257 and bus 125.

44. Referring to claim 43, Morikawa has taught a method comprising:

a) processing data with a hardwired-pipeline circuit. See Fig.4, component 211. This component processes data in the coprocessor. Furthermore, Fig.6 (the bottom portion) shows that the coprocessor is pipelined.

b) loading the processed data into a memory. See Fig.4, and note that after the data is processed, it is loaded into output memory 231.

c) retrieving the processed data from the memory and providing the retrieved processed data to an external source. See column 18, lines 28-32, and Fig.4. The data is retrieved from the memory 231 and sent to the processor via driver 257 and bus 125.

45. Referring to claim 44, Morikawa has taught a method comprising:

a) loading raw data from an external source into a first memory. See Fig.4 and claims 1-2 of Morikawa and note that data from register file 122 in the processor (external source) is received by the coprocessor and loaded into memory portion 230 (input buffer).

b) retrieving the raw data from the first memory. Fig.4 shows that the data is retrieved from the memory portion 230 by the coprocessor processing unit 211.

c) processing the retrieved data with a hardwired pipeline. Once the data has been retrieved, it is processed by the processing unit. See column 18, lines 22-27. Furthermore, Fig.6 (the bottom portion) shows that the coprocessor is pipelined.

d) loading the processed data from the hardwired pipeline into a second memory. See Fig.4, and note that after the data is processed, it is stored in memory portion 231.

e) providing the processed data from the second memory to the external source. See column 18, lines 28-32, and Fig.4. Note that after processing and storage in the second memory, the data is sent to the external processor 201.

46. Referring to claim 45, Morikawa has taught a method as described in claim 44. Morikawa has further taught:

a) loading the raw data comprises loading the raw data via a first port of the first memory. See Fig.4 and note that data is loaded into first memory 230 via its input port.

b) retrieving the raw data comprises retrieving the raw data via the second port of the first memory. See Fig.4 and note that the data is read from memory 230 via its output port and sent to the coprocessor processing unit 211.

c) loading the processed data comprises loading the processed data via a first port of the second memory. See Fig.4 and note that the processed data is written to the second memory 231 via its input port.

d) providing the processed data comprises retrieving the processed data via a second port of the second memory. See Fig.4 and note that when the data is to be sent to the processor 201, it is first retrieved from memory 231 via its output port.

47. Referring to claim 49, Morikawa has taught a method as described in claim 44. Furthermore, Morikawa has inherently taught setting parameters for loading and retrieving the raw data, processing the retrieved data, and loading and providing the processed data. That is, if the system is going to load data from the processor, process the data, and then provide data back to the processor, then the system must set which data will be loaded (i.e., which register the data will be coming from), it must set which operation is to be performed on the data, and it must set which register will be receiving the processed data result. Without these parameters, the functions could not be performed.

48. Referring to claim 1 (under a second interpretation), Morikawa has taught a pipeline accelerator, comprising:

a) a memory. See Fig.4, and note register file 106.

b) a hardwired-pipeline circuit (Fig.4, component 202; also, from the bottom portion of Fig.6, note that the coprocessor is pipelined) coupled to the memory and operable to:

b1) receive data. See Fig.4, and note that coprocessor 202 receives data from saved data storage buffer 232.

b2) load the data into the memory. Fig.4 shows that when the data is received on bus 124 it can be sent to bus 125 via bus 245. Form there, it will be loaded into memory 106.

b3) retrieve the data from the memory. The data will be retrieved by the coprocessor 202 when the register holding the data is identified as being the register to supply data to coprocessor 202.

b4) process the retrieved data. Once the data has been retrieved, it is processed by the coprocessor processing unit 211. See column 18, lines 22-27.

b5) provide the processed data to an external source. See column 18, lines 28-32, and Fig.4. Note that after processing, the data is sent to the processor 201, which is external to the coprocessor (hardwired-pipeline circuit).

49. Claims 1, 4, and 41-43 are rejected under 35 U.S.C. 102(b) as being anticipated by Hennessy and Patterson, "Computer Architecture - A Quantitative Approach, 2nd Edition," 1996 (herein referred to as Hennessy).

50. Referring to claim 1, Hennessy has taught a pipeline accelerator, comprising:

a) a memory. See page 134, Fig.3.4, and note the register file (labeled "Registers").

b) a hardwired-pipeline circuit (the remaining parts of Fig.3.4 excluding the instruction memory and the data memory) coupled to the memory and operable to:

b1) receive data. See page 155, Fig.3.15, and note that when the pipeline circuit executes the "LW R1, B" instruction, data word B will be received from data memory (a load is performed such that B is received).

b2) load the data into the memory. Again, see page 155, Fig.3.15, and note that the "LW R1, B" instruction loads data B into the register file (location R1).

b3) retrieve the data from the memory. See page 155, Fig.3.15, and note that when the pipeline executes the "ADD R3, R1, R2" instruction, the data value B, which was previously stored in register R1, is now retrieved from R1 so that it may be processed by the ADD instruction.

b4) process the retrieved data. The ADD instruction processes the data value B by adding a value to it. The value added to it is the value stored in register R2 (data value C, which was loaded into R2 by the second LW instruction in Fig.3.15).

b5) provide the processed data to an external source. See Fig.3.15 on page 155 and note that after the ADD instruction, the processed data is stored to address A of data memory, which is shown in Fig.3.4 on page 134.

51. Referring to claim 4, Hennessy has taught a pipeline accelerator as described in claim 1. Hennessy has further taught that the pipeline circuit is operable to provide the processed data to the external source by:

a) loading the processed data into the memory. See page 155, Fig.3.15 and note that the ADD instruction, after processing the data B (in R1) will load the result into the register file memory at location R3.

b) retrieving the processed data from the memory and providing the retrieved processed data to the external source. See page 155, Fig.3.15, and note that the SW instruction will retrieve the data from register R3 and provide it to the data memory at address A.

52. Referring to claim 41, Hennessy has taught a method, comprising:

a) loading data into a memory. See page 155, Fig.3.15, and note that the "LW R1, B" instruction loads data B into the register file (location R1).

b) retrieving the data from the memory. See page 155, Fig.3.15, and note that when the pipeline executes the "ADD R3, R1, R2" instruction, the data value B, which was previously stored in register R1, is now retrieved from R1 so that it may be processed by the ADD instruction.

c) processing the retrieved data with a hardwired-pipeline circuit. The ADD instruction processes the data value B by adding a value to it. The value added to it is the value stored in register R2 (data value C, which was loaded into R2 by the second LW instruction in Fig.3.15). Note that Fig.3.4 shows a pipeline.

d) providing the processed data to an external source. See Fig.3.15 on page 155 and note that after the ADD instruction, the processed data is stored to address A of data memory, which is shown in Fig.3.4 on page 134.

53. Referring to claim 42, Hennessy has taught a method as described in claim 41. Hennessy has further taught that the providing the processed data comprises:

a) loading the processed data into the memory. See page 155, Fig.3.15 and note that the ADD instruction, after processing the data B (in R1) will load the result into the register file memory at location R3.

b) retrieving the processed data from the memory and providing the retrieved processed data to the external source. See page 155, Fig.3.15, and note that the SW instruction will retrieve the data from register R3 and provide it to the data memory at address A.

54. Referring to claim 43, Hennessy has taught a method comprising:

a) processing data with a hardwired pipeline circuit. See page 134, Fig.3.4.

b) loading the processed data into a memory. See page 155, Fig.3.15 and note that the ADD instruction, after processing the data B (in R1) will load the result into the register file memory at location R3.

c) retrieving the processed data from the memory and providing the retrieved processed data to the external source. See page 155, Fig.3.15, and note that the SW instruction will retrieve the data from register R3 and provide it to the data memory at address A.

Claim Rejections - 35 USC § 103

55. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

56. Claim 2 is rejected under 35 U.S.C. 103(a) as being unpatentable over Morikawa in view of Tubbs et al., U.S. Patent No. 5,752,071 (herein referred to as Tubbs).

57. Referring to claim 2, Morikawa has taught a pipeline accelerator as described in claim 1 (under the second interpretation). Morikawa has taught that the coprocessor and memory (register file in the processor) are disposed on a single chip (Fig.11), and has not taught that the

memory is disposed on a first integrated circuit and the pipeline circuit is disposed on a second integrated circuit. However, Tubbs has taught that a coprocessor and processor (which includes the claimed memory) are disposed on separate integrated circuits. See column 8, lines 51-52. A person of ordinary skill in the art would have recognized that by disposing the two components on different chips, increased flexibility is realized. For example, if one chip goes bad, it may be replaced without having to replace the other. Or, if an upgrade to one of the components is desired, the chip may be swapped out for an upgraded chip without affecting the other. In addition, as shown in *Nerwin v. Erlichman 168 USPQ 177 (1969)*, to make separable (i.e., to turn a single chip into two chips) is generally not given patentable weight or would have been an obvious improvement. As a result, in order to increase flexibility, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Morikawa such that the memory and pipeline circuit are disposed on separate integrated circuits.

58. Claims 3, 12, and 15 are rejected under 35 U.S.C. 103(a) as being unpatentable over Morikawa in view of FOLDOC.

59. Referring to claim 3, Morikawa has taught a pipeline accelerator as described in claim 1 (under both the first and second interpretations). Morikawa has not taught that the pipeline circuit is disposed on a field-programmable gate array (FPGA). However, FOLDOC has taught that FPGAs are devices which are programmed after manufacture time. See the 1st paragraph of FOLDOC. This is clearly an advantage as designers are able to reprogram/reconfigure the device after it has been made. That is, if any new functionality is desired, a new chip would not have to be manufactured. Instead, the associated functionality would simply be programmed

into the FPGA. Consequently, in order to make the pipeline circuit of Morikawa reconfigurable and, consequently, more flexible, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Morikawa such that the pipeline is disposed on an FPGA as taught by FOLDOC.

60. Referring to claim 12, Morikawa has taught a pipeline accelerator as described in claim 9.

a) Morikawa has not taught that the first and second memories are respectively disposed on first and second integrated circuits. However, as shown in *Nerwin v. Erlichman 168 USPQ 177 (1969)*, to make separable is generally not given patentable weight or would have been an obvious improvement. In this case, it would have been obvious to have the buffers of the coprocessor of Fig.4 to be on separate chips. For instance, everything prior to the processing unit 211 would be on one chip, while the rest would be on another. A person of ordinary skill in the art would have recognized that separating the components into multiple chips allows for more flexibility and cost savings. For example, if the coprocessor were divided into multiple chips and one chip is malfunctioning or is defective, just that chip would need to be replaced while the other functioning chip would be left in tact. This results in not having to spend money replacing the functioning chip. Another advantage would be to gain the ability to upgrade a portion of the coprocessor. As a result, in order to increase flexibility, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Morikawa to include the first and second memories on separate integrated circuits.

b) Morikawa has not taught that the pipeline circuit is disposed on a field-programmable gate array (FPGA). However, FOLDOC has taught that FPGAs are devices which are programmed after manufacture time. See the 1st paragraph of FOLDOC. This is clearly an advantage as

designers are able to reprogram/reconfigure the device after it has been made. That is, if any new functionality is desired, a new chip would not have to be manufactured. Instead, the associated functionality would simply be programmed into the FPGA. Consequently, in order to make the pipeline circuit of Morikawa reconfigurable and, consequently, more flexible, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Morikawa such that the pipeline is disposed on an FPGA, as taught by FOLDOC.

61. Referring to claim 15, Morikawa has taught a pipeline accelerator as described in claim 9.

a) Morikawa has further taught that each of the input-data handler, hardwired pipeline, pipeline interface, and output-data handler has a respective operating configuration. That is, each component in Morikawa operates in a specific fashion and consequently, each component inherently has a respective operating configuration.

b) Morikawa has not taught a configuration manager coupled to and operable to set the operating configurations of the input-data handler, hardwired pipeline, pipeline interface, and output-data handler. However, FOLDOC has taught that FPGAs are devices which are programmed after manufacture time. See the 1st paragraph of FOLDOC. This is clearly an advantage as designers are able to reprogram/reconfigure the device after it has been made. That is, if any new functionality is desired, a new chip would not have to be manufactured. Instead, the associated functionality would simply be programmed into the FPGA. Consequently, in order to make the pipeline circuit of Morikawa reconfigurable and, consequently, more flexible, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Morikawa such that the system is disposed on an FPGA. Furthermore, an FPGA would be coupled to a

configuration manager so that the FPGA may be programmed to include the desired functionality.

62. Claim 3 is rejected under 35 U.S.C. 103(a) as being unpatentable over Hennessy in view of FOLDOC.

63. Referring to claim 3, Hennessy has taught a pipeline accelerator as described in claim 1. Hennessy has not taught that the pipeline circuit is disposed on a field-programmable gate array (FPGA). However, FOLDOC has taught that FPGAs are devices which are be programmed after manufacture time. See the 1st paragraph of FOLDOC. This is clearly an advantage as designers are able to reprogram/reconfigure the device after it has been made. That is, if any new functionality is desired, a new chip would not have to be manufactured. The associated functionality would simply be programmed into the FPGA. Consequently, in order to make the system of Hennessy reconfigurable and, consequently, more flexible, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Hennessy such that the pipeline of page 134 is disposed on an FPGA as taught by FOLDOC.

64. Claims 11 and 46 are rejected under 35 U.S.C. 103(a) as being unpatentable over Morikawa in view of Fette.

65. Referring to claim 11, Morikawa has taught a pipeline accelerator as described in claim 9. Morikawa has not taught a third memory coupled to the hardwired-pipeline circuit, wherein the hardwired pipeline is operable to generate intermediate data while processing the raw data, and wherein the pipeline interface is operable to load the intermediate data into the third memory and

to retrieve the intermediate data from the third memory. However, Fette has taught such a concept. Specifically, Fette has taught that coprocessors contain multiply-accumulate circuitry. See the last sentence of the abstract. A multiply-accumulator is known to include an accumulation register (third memory). With a multiply-accumulate (MAC) operation, values (raw data) are multiplied to generate a multiplication result. This result is then added to the value already in the accumulation register to generate intermediate data. This intermediate data is then stored in the third memory (back into the accumulator), where it is later retrieved and added to the next multiplication result. A MAC operation is a common operation, and it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Morikawa such that the coprocessor performs MAC operations and therefore includes the claimed third memory and its purpose. One would have been motivated to make such a combination to increase the functionality of the coprocessor by giving it MAC functionality.

66. Referring to claim 46, Morikawa has taught a method as described in claim 44. Morikawa has not taught generating intermediate data with the hardwired pipeline in response to processing the raw data, loading the intermediate data into a third memory, and providing the intermediate data from the third memory back to the hardwired pipeline.. However, Fette has taught such a concept. Specifically, Fette has taught that coprocessors contain multiply-accumulate circuitry. See the last sentence of the abstract. A multiply-accumulator is known to include an accumulation register (third memory). With a multiply-accumulate (MAC) operation, values (raw data) are multiplied to generate a multiplication result. This result is then added to the value already in the accumulation register to generate intermediate data. This intermediate data is then stored in the third memory (back into the accumulator), where it is later retrieved and

added to the next multiplication result. A MAC operation is a common operation, and it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Morikawa such that the coprocessor performs MAC operations and therefore includes the claimed third memory and its purpose. One would have been motivated to make such a combination to increase the functionality of the coprocessor by giving it MAC functionality.

67. Claims 13-14 and 47-48 are rejected under 35 U.S.C. 103(a) as being unpatentable over Morikawa in view of Frey.

68. Referring to claim 13, Morikawa has taught a pipeline accelerator as described in claim 9. Morikawa has not taught an input-data queue coupled to the input-data handler and the pipeline interface, wherein the input-data handler is operable to load into the input-data queue a pointer to a location of the raw data within the first memory and wherein the pipeline interface is operable to retrieve the raw data from the location using the pointer. However, Frey has taught such a concept. See Fig.3, components 17 and 21, and column 11, line 67, to column 12, line 5. Essentially, addresses of operands in a multi-operand buffer are stored in an operand fetch queue so that when it is time to fetch the operands, they are located appropriately. A person of ordinary skill in the art would have recognized that by implementing a multi-operand buffer in Morikawa (for buffer 230), multiple operands would be buffered at a time, which ensures that the coprocessor always has enough work to do. Also, if the coprocessor were to stall in any way, the multi-operand buffer would still be able to buffer operands from the processor, thereby allowing the processor to continue on with other work. With such a system, saving the pointer of an operand allows for the location of the operand in the buffer. Clearly, when an instruction is to

operate on an operand, the correct operand should be located, and so the location of the operand must be remembered. Therefore, in order to buffer more operands, ensure work, and avoid stalls, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Morikawa to include a multi-operand input buffer and an input-data queue for holding a pointer to an operand (raw data) within the buffer and then using the pointer to locate the desired operand.

69. Referring to claim 14, Morikawa has taught a pipeline accelerator as described in claim 9. Morikawa has not taught an output-data queue coupled to the output-data handler and the pipeline interface wherein the pipeline interface is operable to load into the output-data queue a pointer to a location of the processed data within the second memory and wherein the output-data handler is operable to retrieve the processed data from the location using the pointer. However, Frey has taught keeping a queue of pointers so that data in a multiple-entry buffer would be appropriately located. See Fig.3, components 17 and 21, and column 11, line 67, to column 12, line 5. A person of ordinary skill in the art would have recognized that by implementing a multi-result buffer in Morikawa (for buffer 231), multiple results would be buffered at a time, which ensures that the coprocessor would function smoothly if bus contention were present. For instance, the regular processor may be trying to write to its own register file. If this is the case, the coprocessor cannot write a result to the register file at the same time. Consequently, if only a single entry output buffer exists, then the result must be stored in that buffer until the register file is available. Until its available, another result cannot be produced as the previous result is already in the buffer. This would result in stalling the coprocessor. With a multiple-entry buffer, the coprocessor would be able to continue executing and storing results

until the register file becomes available. And, with such a system, saving the pointer of the result allows for the location of the result in the buffer. Clearly, when a result is to be written to the register file, the correct result should be written, and so the location of the result must be remembered. Therefore, in order to avoid stall potential, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Morikawa to include a multiple-entry output buffer and an output-data queue for holding a pointer to a result (processed data) within the buffer and then using the pointer to locate the desired data.

70. Referring to claim 47, Morikawa has taught a method as described in claim 44. Morikawa has not taught loading into an input-message queue a pointer to a location of the raw data within the first memory, and wherein retrieving the raw data comprises retrieving the raw data from the location using the pointer. However, Frey has taught such a concept. See Fig.3, components 17 and 21, and column 11, line 67, to column 12, line 5. Essentially, addresses of operands in a multi-operand buffer are stored in an operand fetch queue so that when it is time to fetch the operands, they are located appropriately. A person of ordinary skill in the art would have recognized that by implementing a multi-operand buffer in Morikawa (for buffer 230), multiple operands would be buffered at a time, which ensures that the coprocessor always has enough work to do. Also, if the coprocessor were to stall in any way, the multi-operand buffer would still be able to buffer operands from the processor, thereby allowing the processor to continue on with other work. With such a system, saving the pointer of an operand allows for the location of the operand in the buffer. Clearly, when an instruction is to operate on an operand, the correct operand should be located, and so the location of the operand must be remembered. Therefore, in order to buffer more operands, ensure work, and avoid stalls, it

would have been obvious to one of ordinary skill in the art at the time of the invention to modify Morikawa to include a multi-operand input buffer and an input-message queue for holding a pointer to an operand (raw data) within the buffer and then using the pointer to locate the desired operand.

71. Referring to claim 48, Morikawa has taught a method as described in claim 44. Morikawa has not taught loading into an output-message queue a pointer to a location of the processed data within the second memory, and wherein retrieving the processed data comprises retrieving the processed data from the location using the pointer. However, Frey has taught keeping a queue of pointers so that data in a multiple-entry buffer would be appropriately located. See Fig.3, components 17 and 21, and column 11, line 67, to column 12, line 5. A person of ordinary skill in the art would have recognized that by implementing a multi-result buffer in Morikawa (for buffer 231), multiple results would be buffered at a time, which ensures that the coprocessor would function smoothly if bus contention were present. For instance, the regular processor may be trying to write to its own register file. If this is the case, the coprocessor cannot write a result to the register file at the same time. Consequently, if only a single entry output buffer exists, then the result must be stored in that buffer until the register file is available. Until its available, another result cannot be produced as the previous result is already in the buffer. This would result in stalling the coprocessor. With a multiple-entry buffer, the coprocessor would be able to continue executing and storing results until the register file becomes available. With such a system, saving the pointer of the result allows for the location of the result in the buffer. Clearly, when a result is to be written to the register file, the correct result should be written, and so the location of the result must be remembered. Therefore, in

order to avoid stall potential, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Morikawa to include a multiple-entry output buffer and an output-message queue for holding a pointer to a result (processed data) within the buffer and then using the pointer to locate the desired data.

72. Claims 16 and 50 are rejected under 35 U.S.C. 103(a) as being unpatentable over Morikawa.

73. Referring to claim 16, Morikawa has taught a method as described in claim 9.

a) Morikawa has inherently taught that each of the input-data handler, hardwired pipeline, pipeline interface, and output-data handler has a respective operating status. That is, at the very least, each component in the system is either operating or not operating (and these are statuses).

b) Morikawa has not taught an exception manager coupled to and operable to identify an exception in the input-data handler, hardwired pipeline, pipeline interface, or output-data handler in response to the operating statuses. However, Official Notice is taken that checking for errors in a pipeline during processing is well known and accepted in the art. For instance, if a coprocessor is performing a divide operation, which is a well known operation performed by coprocessors, then a division of a number by 0 should raise an exception, as it is an illegal operation. This type of error should be monitored so that the system can take appropriate action to correct the error. As a result, in order to ensure proper execution, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Morikawa such that the coprocessor of Morikawa performs divide operations (in order to obtain the ability to perform division) and exceptions are detected in a hardwired pipeline.

74. Referring to claim 50, Morikawa has taught a method as described in claim 44. While Morikawa has not explicitly taught determining whether an error occurs during the loading and retrieving of the raw data, the processing of the retrieved data, and the loading and providing of the processed data, Official Notice is taken that checking for errors during processing is well known and accepted in the art. For instance, if a coprocessor is performing a divide operation, which is a well known operation performed by coprocessors, then a division of a number by 0 should raise an error as it is an illegal operation. This type of error should be monitored so that the system can take appropriate action to correct the error. As a result, in order to ensure proper execution, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Morikawa such that the coprocessor of Morikawa performs divide operations (in order to obtain the ability to perform division) and errors are detected during the loading and retrieving of the raw data, the processing of the retrieved data, and the loading and providing of the processed data.

Conclusion

75. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure. Applicant is reminded that in amending in response to a rejection of claims, the patentable novelty must be clearly shown in view of the state of the art disclosed by the references cited and the objections made. Applicant must also show how the amendments avoid such references and objections. See 37 CFR § 1.111(c).

Lewis, U.S. Patent No. 6,624,819, has taught a method and system for providing a flexible and efficient processor for use in a graphics processing system. Specifically, a

coprocessor includes two buffers in which data is loaded and retrieved and stored in again for access by an external source.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to David J. Huisman whose telephone number is (571) 272-4168. The examiner can normally be reached on Monday-Friday (8:00-4:30).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

DJH
David J. Huisman
October 3, 2006